Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2023-1

15 de junio de 2023
Actividad

Actividad 5

Iterables

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/A5/`
- **Hora del *push*:** 16:40

Importante: Antes de comenzar, comprueba que Git este funcionando correctamente en tu repositorio privado. Para esto, **sube los archivos base de la actividad de inmediato** (*add*, *commit*, *push*). Se espera que en esta actividad (así como en las demás actividades y tareas) utilices Git a lo largo de **todo tu desarrollo** como una herramienta, no sólo como un método de entrega. Es por esto que recomendamos enfáticamente que vayas subiendo tus cambios constantemente (***push***), ya que **problemas de último minuto** relacionados con la entrega y Git **no serán considerados**.

Introducción

Después del éxito innegable del DCChain[1], el DCC decide abrir el primer supermercado cibernético: **DCCybermercado**. Deberás ayudar al DCC a modelar este supermercado, permitiéndoles cargar la información de sus productos, realizar consultas sobre los productos y, finalmente, manejar los carritos de compras.

Archivos

En el directorio de la actividad encontrarás los siguientes archivos:

Archivos de datos

- No modificar `productos.csv`: Este archivo contiene la información de los productos disponibles. El formato del archivo es:

  ```
  id_producto,nombre,precio,pasillo,medida,unidad_medida
  ```

  donde `id_producto`, `precio` y `medida` corresponden a números enteros.

- No modificar `categorias.csv`: Este archivo contiene la información de todas las categorías de los productos del archivo anterior. Un mismo producto, puede estar relacionado a una o más categorías. El formato del archivo es:

  ```
  nombre_categoria,id_producto
  ```

[1] Para un mayor contexto, revisa la experiencia de la semana 9.

donde `id_producto` corresponden a un número entero.

Archivos de código

- Modificar `funciones.py`: Contiene las funciones necesarias para cargar y manejar la información.
- No modificar `utilidades.py`: Contiene la definición de *namedtuples* y funciones necesarias para cargar y manejar la información.
- No modificar `test.py`: Contiene el código necesario para ejecutar los *tests* relacionados a las funcionalidades.

Flujo del programa

Esta actividad consta de tres partes, en las cuales se te pedirá que implementes funciones que permitan cargar información, realizar consultas y manejar un carro de compras.

1. Parte 1 – Cargar datos

Para que puedas implementar correctamente las funcionalidades, te entregamos las siguientes *namedtuples* **ya implementadas** en el módulo `utilidades.py`:

- No modificar `Producto`: Posee los atributos `id_producto (int)`, `nombre (str)`, `precio (int)`, `pasillo (str)`, `medida (int)` y `unidad_medida (str)`.
- No modificar `Categoria`: Posee los atributos `nombre_categoria (str)` y `id_producto (int)`.

Para cargar los datos y utilizar las *namedtuples* anteriores, deberás completar las siguientes funciones del archivo `funciones.py`:

- Modificar `def cargar_productos(ruta: str) -> Generator:`
  Esta función generadora recibe un `str` con la ruta del archivo y va retornando instancias de `Producto` según el contenido del archivo.
- Modificar `def cargar_categorias(ruta: str) -> Generator:`
  Esta función generadora recibe un `str` con la ruta del archivo y va retornando instancias de `Categoria` según el contenido del archivo.

2. Parte 2 – Consultas

Para poder manejar el **DCCybermercado**, deberás completar una serie de consultas que trabajarán sobre los datos cargados.

Importante Se encuentra **estrictamente prohibido** el uso de: ciclos `for` y `while`; estructuras de datos `list`, `tuple`, `dict`, `set`; cualquier tipo de estructura creada por comprensión; y otras librerías a las dadas en los archivos base.

Para lograr implementar las funciones anterior, ten en mente el uso de funciones `lambda`, pero considera que el énfasis está en el uso de las funciones `map`, `filter` y `reduce`.

Las funciones a completar en el archivo `funciones.py` son:

- Modificar `def obtener_productos(generador_productos: Generator) -> map:`
  Recibe un generador con instancias de `Producto` y retorna un objeto de tipo `map` con el nombre de todos los productos, en el mismo orden que venían en el generador.

- Modificar `def obtener_precio_promedio(generador_productos: Generator) -> int:`
  Recibe un generador con instancias de `Producto` y calcula el precio promedio de estos productos. El precio obtenido debe ser transformado a un número entero utilizando la función `int()` y retornado.

  Para obtener una copia de un generador, puedes utilizar la función:

  - No modificar `def duplicador_generadores(generador: Generator) -> tuple:`
    Recibe un generador y retorna una tupla con dos copias del mismo generador.

- Modificar `def filtrar_por_medida(generador_productos: Generator, medida_min: float, medida_max: float, unidad: str) -> filter:`
  Recibe un generador con instancias de `Producto` y retorna un objeto de tipo `filter` con los productos filtrados.

  Los productos se filtran de forma que su unidad de medida corresponda a la indicada en `unidad` y que su medida se encuentre entre los valores dados por `medida_min` y `medida_max`, inclusive.

- Modificar `def filtrar_por_categoria(generador_productos: Generator, generador_categorias: Generator, nombre_categoria: str) -> filter:`
  Recibe un generador con instancias de `Producto`, un generador con instancias de `Categoria` y el nombre de una categoría. Retorna un objeto de tipo `filter` con todos los productos correspondientes a la categoría indicada.

  Los pasos para filtrar los productos, son los siguientes:

  1. Encontrar los *ids* de todos los productos pertenecientes a la categoría indicada.
  2. Se transforma el generador obtenido en el paso anterior, en una lista utilizando la función:
     - No modificar `def generador_a_lista(generador: Generador) -> list:`
       Recibe un generador y retorna una lista con el contenido de dicho generador.
  3. Filtra los productos correspondientes a la lista de *ids*.

- Bonus Modificar `def agrupar_por_pasillo(generador_productos: Generator) -> groupby:`
  Recibe un generador con instancias de `Producto` y retorna un objeto de tipo `groupby`, donde se agrupan los productos en base a su pasillo. Para lograr lo anterior, deberás investigar y utilizar la función `groupby` de la librería `itertools`. Puedes asumir que los productos vendrán ordenados por pasillo.

  Por ejemplo, si se utiliza un generador con los siguientes productos:

```
6,Cebollas,790,Pasillo 3,500,gr
7,Papas,1490,Pasillo 3,1000,gr
8,Jamón,2990,Pasillo 4,200,gr
9,Pan de molde,1490,Pasillo 4,500,gr
10,Salsa de tomate,990,Pasillo 5,500,ml
```

  Después de agruparlos por pasillo, el objeto de tipo `groupby` contendrá la siguiente información:

```
Pasillo 3 : [Cebollas, Papas]
Pasillo 4 : [Jamón, Pan de molde]
Pasillo 5 : [Salsa de tomate]
```

3. Parte 3 – Carrito de compras

En esta última parte, deberás aplicar tus conocimientos de **iterables personalizados**, para implementar la clase `IteradorCarrito`, que corresponde al iterador de la clase `Carrito`.

Ambas clases se encuentran en el archivo `funciones.py` y la intención de estas es poder recorrer los productos —de forma ordenada y ascendente— según su precio. Para lograr lo anterior, deberás completar los métodos faltantes de las clases `Carrito` e `IteradorCarrito`:

- `class Carrito`: Esta clase recibe como argumento una lista de productos.
  - Modificar `def __iter__(self)`: Este método retorna una instancia `IteradorCarrito` con los datos correspondientes.
- `class IteradorCarrito`: Esta clase recibe como argumento una instancia de lista de productos y guarda una copia de ésta como atributo (`lista_productos`).
  - Modificar `def __iter__(self)`: Este método debe retornar a la instancia misma del iterador (`self`).
  - Modificar `def __next__(self)`: Este método es el encargado de encontrar el **siguiente producto con menor precio**. En caso de que no hayan más elementos que retornar, se levanta la excepción correspondiente de este iterador.

Notas

- Recuerda que la ubicación de tu entrega es en tu **repositorio personal**. Verifica que no estés trabajando en el **Syllabus**.
- Recuerda que esta evaluación presenta corrección **automatizada**. Si entregas un código que se cae al momento de correr los *tests*, será evaluado con 0 puntos.
- Se recomienda completar la actividad en el orden del enunciado, a excepción del Bonus.
- El resolver completamente el bonus y pasar todos los *tests* asociados, permite obtener 1 punto adicional.
- Si aparece un error inesperado, ¡léelo! Intenta interpretarlo y/o buscarlo en Google.

Objetivo de la actividad

- Implementar una función generadora, utilizando correctamente `yield`.
- Aplicar conocimientos de iterables utilizando funciones `map`, `filter` y `reduce`.
- Crear un iterable y un iterador personalizado definiendo correctamente los métodos `__iter__` y `__next__`.